Set-up for Python Data Analysis

Open a notebook, either a Jupyter Notebook on your computer from the Anaconda download or a Colab Notebook on Google Colab. Then do the following, adapted to a specific analysis.

Python consists of the core language plus many packages that extend the core language with additional functions. However, most of the analyses we pursue are *not* part of the core language, so we first import packages of needed functions before analysis begins.

Time Stamp

Not needed, but useful to know the time and date at which you conducted the analysis. The `now()` function that provides this information is not part of core Python, but is available from the `datetime` package, abbreviated here as `dt`. The `datetime` package is not included with the original Python distribution, so separately access with the `import` method.

```
from datetime import datetime as dt
now = dt.now()
print ("Analysis on", now.strftime("%Y-%m-%d"), "at", now.strftime("%H:%M"))
```

```
Analysis on 2021-06-20 at 09:07
```

Refer to the package with its designated abbreviation `dt`. To run the `now()` function, refer to it as `dt.run()`, that is, the package name or abbreviation, a period, then the function name. This convention lets Python know where to locate the corresponding function that is not included with the original Python distribution.

Import Data Analysis Packages

The core language itself does not contain specialized data analysis functions, so first reference the packages that contain these functions before doing data analysis.

> Every data analysis invokes functions from at least two packages, `pandas` and `numpy`.

- `pandas`, a general package for reading, storing, and manipulating data in data frames, standard row x column data tables
- `numpy`, a general package for numerical computations, upon which many other packages rely, including `pandas`

Data analysis usually involves some form of data visualization as well.

> Find the basic data visualization functions in the `matplotlib` package, with increasing reliance upon the more elegant `seaborn`, which is based on `matplotlib`.

- `matplotlib`, basic plotting library for data visualizations
- `seaborn`, more elegant, higher-level plotting library based on `matplotlib`

Access these packages with the `import` method. The optional `as` parameter provides an abbreviation from which to refer to the package functions.

In this Jupyter Notebook, we only need the `pandas` package for reading a data file, but typically reference all four packages in a complete data analysis. The abbreviations listed are traditional for data analysis with Python.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Read and Display Data

There are two different situations in which the contents of data files are read into a working Python program. Read the data from ...

- the web
- a file on a user's file directory, either locally or networked

To read data files from the web, you just need to specify the file URL (web address) when calling a read function. This section explains how to show the default location of reading and writing data files on your local or related computer file system.

To read and write data files, each notebook begins with a default location.

> *Current working directory*: Folder (directory) in your computer's file system that is the default location where Python reads and writes data files.

To read and write data files, the concept of a current working directory allows simplified references to data files without the tedium and error-prone process of listing the full path name from the root level of your computer, such as C: on Windows. Instead, the files are reference relative to the folder (directory) from which the Python program is being run.

File Directory

On Your Computer

Before opening a Jupyter Notebook, first create a folder to store all notebooks and data files. A suggestion is a folder called Python in your Documents folder. The first screen after opening the Jupyter Notebook app is a listing of folders and files. Click on the `Documents` folder and then the Python folder to begin analysis from that location.

Path names that specify where to read a file and write a file begin with the current working directory as the "home" location. For example, it may be convenient to store your data files in a folder named data, located at the top-level of your current working directory. Then all path names to locate a data file stored in this data folder begin with data. The forward slash indicates a sub-folder, that is, sub-directory.

List the current working directory with the `getcwd()` function from the os package.

```
import os
os.getcwd()
```

```
'/content'
```

This example was run on Google Colab, described next, so the current directory has neither a Windows or Macintosh file reference.

On Google Colab

Colab is Google's free cloud computing enviornment. All that is needed to access is a Google account.

Colab Notebooks, essentially Jupyter Notebooks, do not access the files on your personal computer. Instead, they use Google Drive as your file directory. For convenience, create a folder called `data` on your Google Drive and copy one or more data files to that location. Colab will automatically create a folder called `Colab Notebooks` on your Google Drive to store your Jupyter Notebook files.

To access your Google Drive files requires the use of the `mount()` function in the `google.colab` package, imported here with the abbreviation of `drive`, so the function reference is `drive.mount()`. Mount your Google Drive in the `content` directory (folder) and the nested directory called `drive`. When you request to mount the drive you will be provided a link to obtain an authorization code.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Mange the files on your Google Drive, such as storing a data file in a folder called `data`.

To read or write files to or from Google Drive, also include `\MyDrive` in the path name because that reference is to your Google Drive. The full reference to your Google Drive essentially your current working directory:

'/content/drive/MyDrive/'

If you have a folder on the top level of `MyDrive` called `data`, and a file named `employee.xlsx` in that folder, here is the full path name to locate that file:

'/content/drive/MyDrive/data/employee.xlsx'

Note the path name is included with quotes.

Once `drive` is mounted so that you can access the data file, as well as any other existing files, you can also view the path name for any file in the system. Click on the folder icon in the extreme left margin to explore your `drive` file directory (folder). Click on `MyDrive` to access your Google Drive files. To can locate any specific file in that directory, hover the mouse over it, and then click on the three displayed dots. From the displayed choices, choose `Copy path` to get the exact path name to reference that specific file.

Read

The pandas package defines the concept of a data frame and provides the needed functions for reading `csv` or Excel data files (or many other formats) into a data frame. You are free to use any valid name for the data frame, but recommend to read data into a data frame named *d*. The *d* stands for data, entered with only one keystroke. Within a Python analysis, reference the data frame, not the file from which the data were read.

Read an Excel data file from your local file system into the `pandas` data frame named *d* with the `pandas` function `read_excel()`. Comment out other lines of code with the Python (and R) comment symbol: `#`.

- In this example, run on Google Colab, the uncommented read statement is for the Excel data file stored in the `data` folder created on Google Drive.
- The first commented read statement is to read the file on the web.
- The second commented read statement is for the same Excel data file, but for Python running on the user's computer, stored on the user's `data` folder relative to folder that contains the running Python notebook.

```
d = pd.read_excel('/content/drive/MyDrive/data/employee.xlsx')
#d = pd.read_excel('http://web.pdx.edu/~gerbing/data/employee.xlsx')
#d = pd.read_excel('data/employee.xlsx')
```

Display

After reading data into a data frame, *always* check what was just read. Here, get the dimensions of the corresponding data frame and look at the first five rows of data.

The dot notation, the period after the *d* indicates that the corresponding reference to `shape` is for the *d* data frame. Each instance of the data frame object, here *d*, has an attribute (property) called `shape`. Here retrieve the shape of *d*, rows by columns.

```
d.shape
```

```
(37, 9)
```

Most references to pre-programmed functions provide the option of passing specific parameter values within parentheses. But even if all the default values are accepted, the parentheses must still be provided, even if empty. The `head()` function has only one parameter, `n`, which is the number of rows of data to display. The default value is five. So following function calls provide the same result:

- head()
- head(5)
- head(n=5)

```
d.head()
```

	Name	Years	Gender	Dept	Salary	JobSat	Plan	Pre	Post
0	Ritchie, Darnell	7.0	M	ADMN	53788.26	med	1	82	92
1	Wu, James	NaN	M	SALE	94494.58	low	1	62	74
2	Hoang, Binh	15.0	M	SALE	111074.86	low	3	96	97
3	Jones, Alissa	5.0	F	NaN	53772.58	NaN	1	65	62
4	Downs, Deborah	7.0	F	FINC	57139.90	high	2	90	86

Note that unfortunately (in my opinion) Python starts row numbering (and everything else) with 0 instead of 1. For example, the row of data numbered 4 is the fifth row of data.

Also note that in the corresponding Excel file with the data read into the *d* data frame, the cell for variable *Years* for James Wu is missing, it is blank. When read into the data frame, the corresponding cell displays the missing data code for `pandas`, `NaN`, abbreviated from Not a Number.

Data Types

Computers store different types of data values differently. The biggest distinction is between numeric and non-numeric data. With numeric data, further distinguish between data values that have no decimal digits, integers, and those with decimal digits (even if all 0's), floating point values.

> *Data type*: The way in which data values are stored in computer memory.

Other data types exist, but in this example, read each variable into one of three Python data types:

- integer (int64) for 64-bit integers, can be very, very large numbers
- floating-point (float64) for potentially very large numbers with decimal digits
- object for non-numeric values

When working with data frames, organize data by variables (columns). Show the type of each variable with the `info()` function. In Python, methods and functions generally perform the same purpose, but methods are specified without trailing parentheses.

```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37 entries, 0 to 36
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Name    37 non-null     object 
 1   Years   36 non-null     float64
 2   Gender  37 non-null     object 
 3   Dept    36 non-null     object 
 4   Salary  37 non-null     float64
 5   JobSat  35 non-null     object 
 6   Plan    37 non-null     int64  
 7   Pre     37 non-null     int64  
 8   Post    37 non-null     int64  
dtypes: float64(2), int64(3), object(4)
memory usage: 2.7+ KB
```

We see, for example, that *Name* is a variable that has non-numeric characters as data values. *Salary* is a variable with data values that have decimal digits, and *Pre* is a variable with integer data values. For some reason, the variable *Years* also only has integer values, but is represented as a `float64` variable, that is, data values with decimal digits. Perhaps the missing data value, 36 instead of 37 data values, triggered that designation.